Chat server in C

Raúl Rosario Sandoval Galaviz

Instituto Tecnológico y de Estudios Superiores de Monterrey

A00232156

Abstract

A chat server usually refers to a kind of communication over a network that allows real time transmission of messages from sender to receiver. This project has the purpose to create a chat server using concurrency and threads. Along this document, we will find different references that were used to make this chat service possible, concept like threads and sockets will be explained, alongside the concurrency versus parallelism distinction and the structure of the solution to the presented problem.

1. Context of the problem.

1.1 A little bit of history.

The earliest form of a live chat software was conceived at the university of Illinois in the 1970s, in this service multiple users were able to work on a single text document, yes just like google docs! However, the real concept of an online chat was conceived until 1973, and Doug Brown, came up with the idea of a program that would allow group chats. This software was named "Talkomatic", this software was able to host up to five people at the same time, making it the first live chat service in history [1]. This concept evolved slowly to what we know as chat services nowadays, by the 2000s programs like **ICQ Chat** allowed a desktop client anonymous communication. Chat rooms' popularity grew as **Yahoo, AOL, MSN, and Google** also got into the game in the **90s and early 2000s,** sending text messages was smooth, something not that common during the time, considering that the dialup internet average speed in 2003 was around 56 kbps [2]. Finally, at the end of the 2000s and early 2010s apps like **WhatsApp, WeChat, LINE and Telegram** launched, allowing users to have easy access to a chat service via their smartphones.

1.2 Why a local chat server?

Today, we take internet access for granted, but in areas where internet availability is inexistent just like the author's hometown, *Huatabampo*, multiple businesses are based in rural areas, mostly because its main natural resource is farmlands [5], where internet connection is not yet available or possible. [4]. So, in justification for this project, a way of transmitting information or messages via a local area network could be a helpful tool available for this kind of businesses since internet connection is not necessary for this kind of connections [9]. Mobile networks are also an unreliable option, for reference image 1.1 shows a representation of Telcel's mobile network coverage, of course, this is obviously an estimation made by Telcel, and the reality of the coverage quality and actual range is different. On the other hand, Internet coverage is even less likely, as we can see in image 1.2 internet is not exactly an option.

Image 1.1 In yellow, not guaranteed 4G coverage, in green 4G coverage.[3]

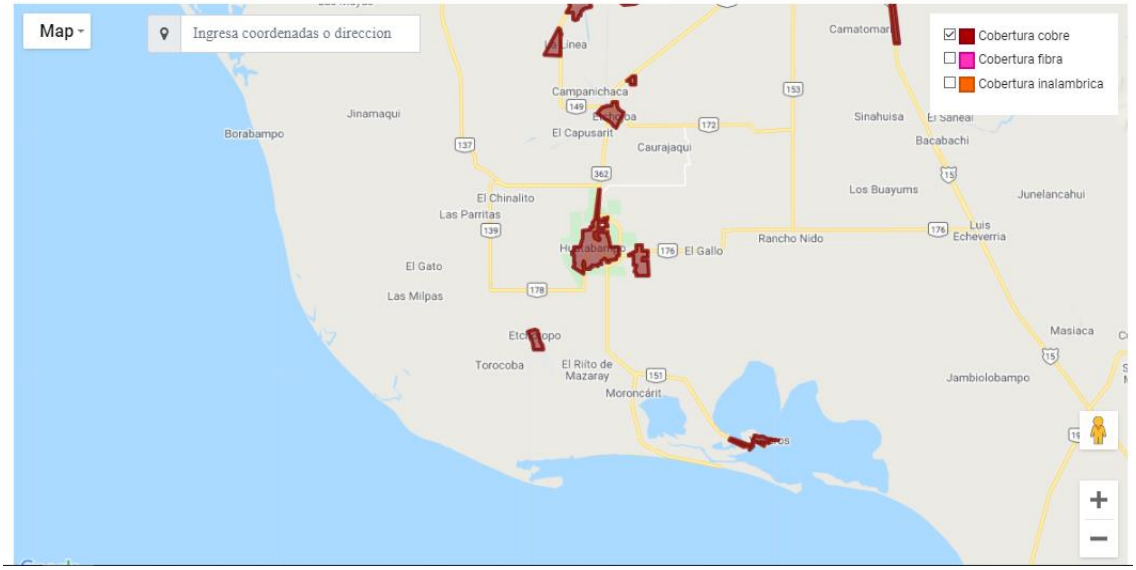


Image 1.2 In red, copper internet coverage in the Huatabampo municipality. [4]

2. Concurrency

2.1 What is Concurrency?

Concurrency means multiple computations are happening at the same time. Today, concurrency is everywhere in modern programming. For example:

- Multiple computers in a network
- Multiple applications running on one computer
- Multiple processors in a computer (today, often multiple processor cores on a single chip) [6]

As we read above, some of these functions are taken for granted by users, downloading files while listening to music, or maybe working on a text file while surfing the internet. Even the simplest of applications are required to do multiple things at a time. [7].

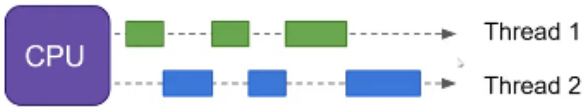


Image 2.1. Concurrency using threads. We can see how the CPU switches between tasks, executes one of them for a time and then switches again. [10]

2.1.1 Concurrency strategies.

There 2 common strategies to follow for concurrent programming:

Sharing memory:

Two programs interact by reading and writing to shared objects in memory [6].

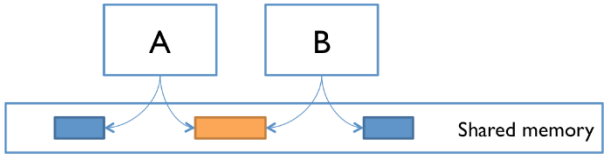


Image 2.2 Two programs or processes sharing an object.

Message passing

In the message-passing model, concurrent programs interact by sending messages to each other through a communication channel [6]

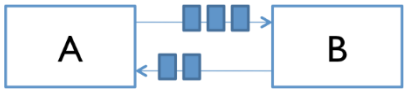


Image 2.3 Two programs or processes passing messages between each other.

2.1.2 Processes & threads

The previous strategies are about **how concurrent programs communicate**, however, concurrent programs also come in two different **kinds**: processes and threads.

2.1.2.1 Process.

A process means any program is in execution. Controlling processes contains information about the process itself, like process priority, process id, process state, CPU, register, etc. Processes do not share memory - at least in an easy way - with any other processes since they are isolated from each other. [9]

2.1.2.2 Thread.

A thread is a different story, is a segment of a process, meaning a single process can have multiple threads. They do not isolate and takes less time to terminate. And is more efficient in term of communication. A thread has 3 states: running, ready, and blocked. [9]

3 Concurrency vs parallelism.

3.1 Why is a chat server and client NOT parallelism?

In simple terms, parallelism is the process of **splitting a single task into subtasks.** This means, that every thread or process is going to be assigned a part of the problem and will solve only that part. [10]

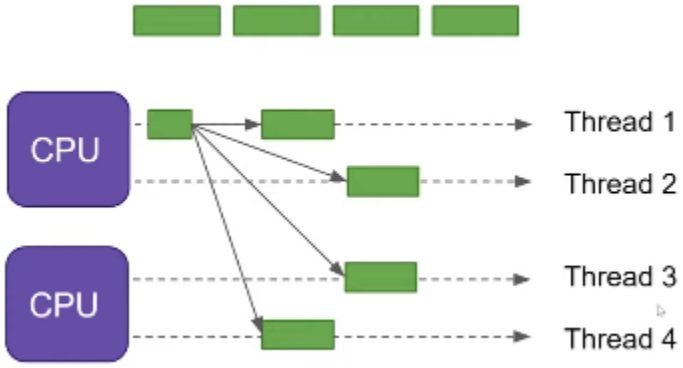


Image 3.1: A parallelism diagram.[10]

4 Solution.

4.1 Threading and concurrency Paradigm.

As we mentioned before, the best way to solve this problem is by using concurrency, since we are not solving a single problem at the same time nor are we assigning a slice of the problem to each thread.

4.2 GUI with GTK3 and GLADE

During this project, GTK was often used as a way of providing a GUI in C for Linux systems. GTK is an open-source widget toolkit, that allows free and paid software to use it. [12] Why would I choose this toolkit in particular? After a long investigation, trying to find the best way to create GUIs with C, I managed to get multiple proposals [14]:

-Turbo C: Absolutely not, it is a discontinued IDE that has been irrelevant and outdated since 1990, older than I AM! Already replaced with Turbo C++, another discontinued IDE. Based on that information. I chose to look for other options.[16]

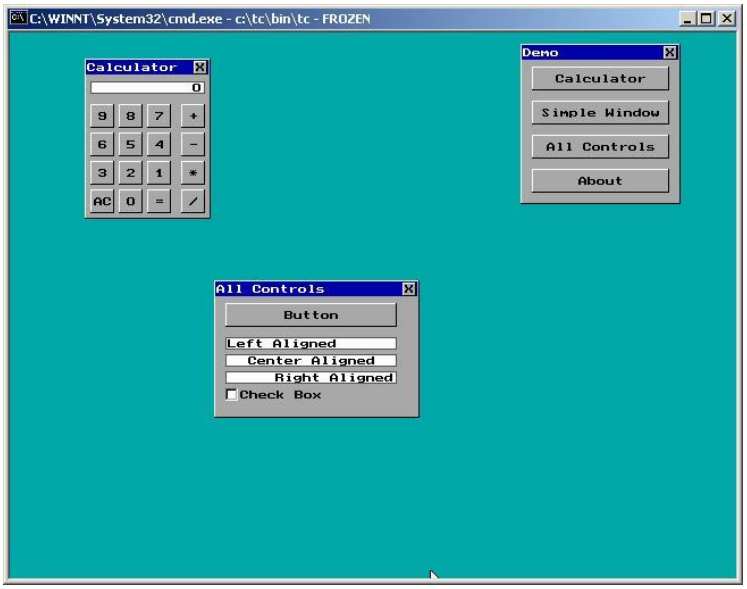


Image 4.1 Turbo C could create this kind of interfaces, not bad, for Windows 95!

Another possibly laid to me was doing it with GTK, obviously, nobody mentioned Glade, a great tool that enables easy development of user interfaces for the GTK toolkit [16]. The user interfaces created inside the program are saved as an XML file which is later used to define your interface inside a C program, not only it creates Widgets, but it also creates signal responses to said widgets, for example, a button could be click,

hovered, released, etc. And glade allows us to define out handler functions right away; it is up to us to decide what to do with the signal. [16].

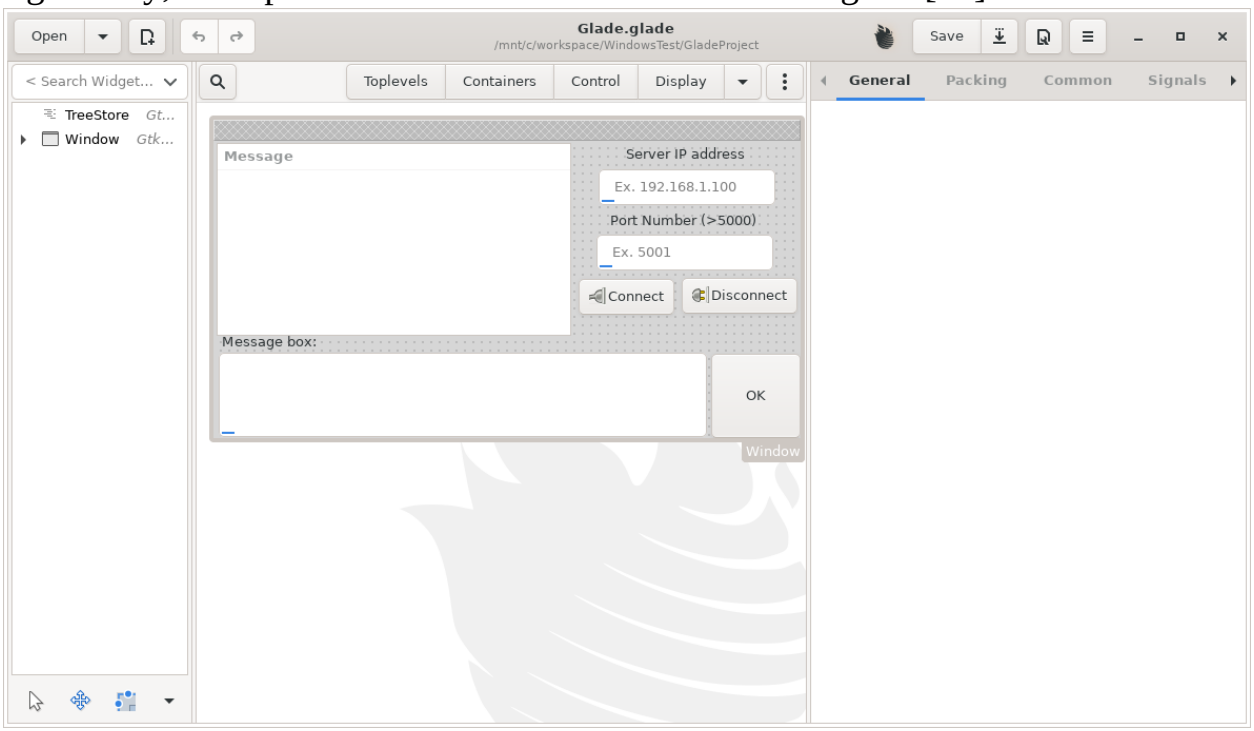


Image 4.2. This is the .glade file generated for this project.

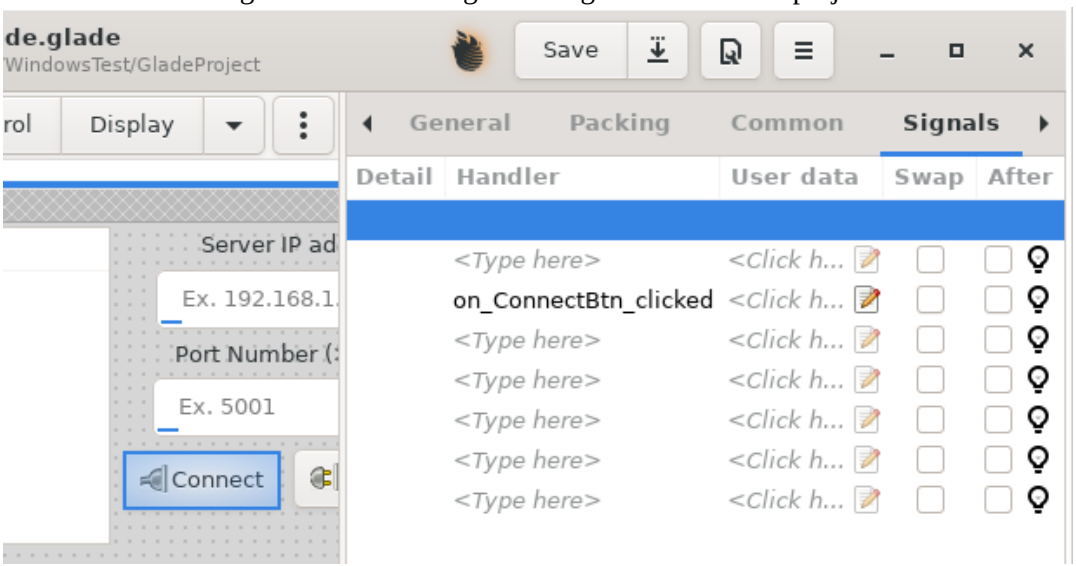


Image 4.3. A button with it's different possible signals and Handlers.

The process of creating a GUI inside GLADE is based on Hierarchy , each individual element, called in GTK "widget" has a position inside the GTK hierarchy, mentioning some of this examples, a GTKWidget element is a generic way of creating a widget, however each of this widgets are different,

for example, GTKWindow, allows us to create a blank window in order to work within it, in the hierarchy, it's one of the highest level Widgets, allowing it to contain a great number of elements inside it [13].

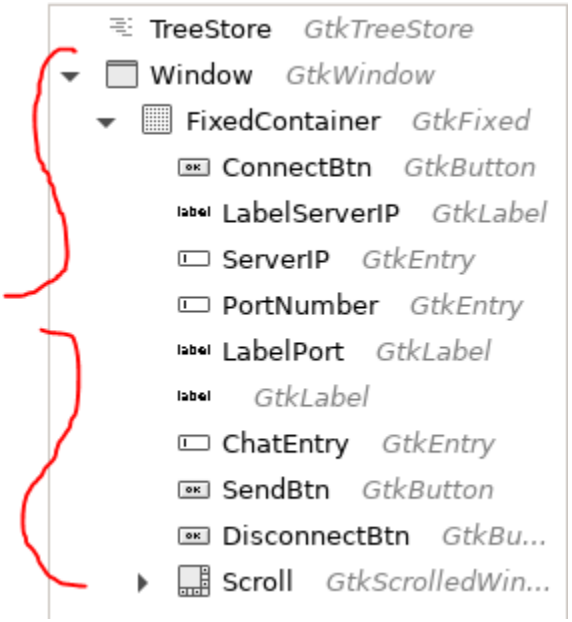


Image 4.4 Hierarchy in GTK3.0 and glade.

4.3 Tying GUI up with a C program.

In my implementation, first, defining the whole interface was easy thanks to Glade, as we mention before, Glade creates a XML file that can be used with the function **gtk_builder_new_from_file (const gchar *filename)** from the GTK 3.0 toolkit. After that, using GTK functions to tie everything up to Widgets our GUI.

```
GtkWidget *window;
GtkWidget *FixedContainer;
GtkWidget *ConnectBtn;
GtkWidget *DisconnectBtn;
GtkWidget *SendBtn;
GtkWidget *LabelServerIp;
GtkWidget *LabelPortNumber;
GtkWidget *EntryServerIp;
GtkWidget *EntryPortNumber;
GtkWidget *ChatEntry;
GtkTreeStore *TreeStore;
GtkTreeView *TreeView;
GtkTreeViewColumn *cx1;
GtkCellRenderer *cr1;
GtkTreeSelection *selection;
GtkBuilder *builder;
GtkTreeIter iter;
```

Image 4.4 Creating every single widget from our application as a global variable to be able to use it everywhere in our code.

```
builder = gtk_builder_new_from_file("./GladeProject/Glade.glade");
//Creates widget window, associated with the "Window" property of our XML File
window = GTK_WIDGET(gtk_builder_get_object(builder,"Window"));
//Create our "X" signal, in order to destroy the window
g_signal_connect(window,"destroy",G_CALLBACK(gtk_main_quit),NULL);

//Connect GLADE established signals to C file Widgets
gtk_builder_connect_signals(builder,NULL);
FixedContainer =  GTK_WIDGET(gtk_builder_get_object(builder,"FixedContainer"));

ConnectBtn = GTK_WIDGET(gtk_builder_get_object(builder,"ConnectBtn"));

DisconnectBtn = GTK_WIDGET(gtk_builder_get_object(builder,"DisconnectBtn"));

gtk_widget_set_sensitive (DisconnectBtn, FALSE);

SendBtn = GTK_WIDGET(gtk_builder_get_object(builder,"SendBtn"));

gtk_widget_set_sensitive (SendBtn, FALSE);
```

Image 4.5 Tying up everything in C to what was created in Glade. Signals, Widgets and Names are all associated via our GTKbuilder. [18]

```
//Show every widget created, even the window!
gtk_widget_show_all(window);
//Creates the main program loop.
gtk_main();
```

Image 4.6 in order to be able to see the widget (including the window itself, since it is a widget) we have to make use of gtk_widget_show_all(window) and gtk_main(). [18]

All of this should leave us with a GUI with every element that we defined in our glade program (image 4.2).

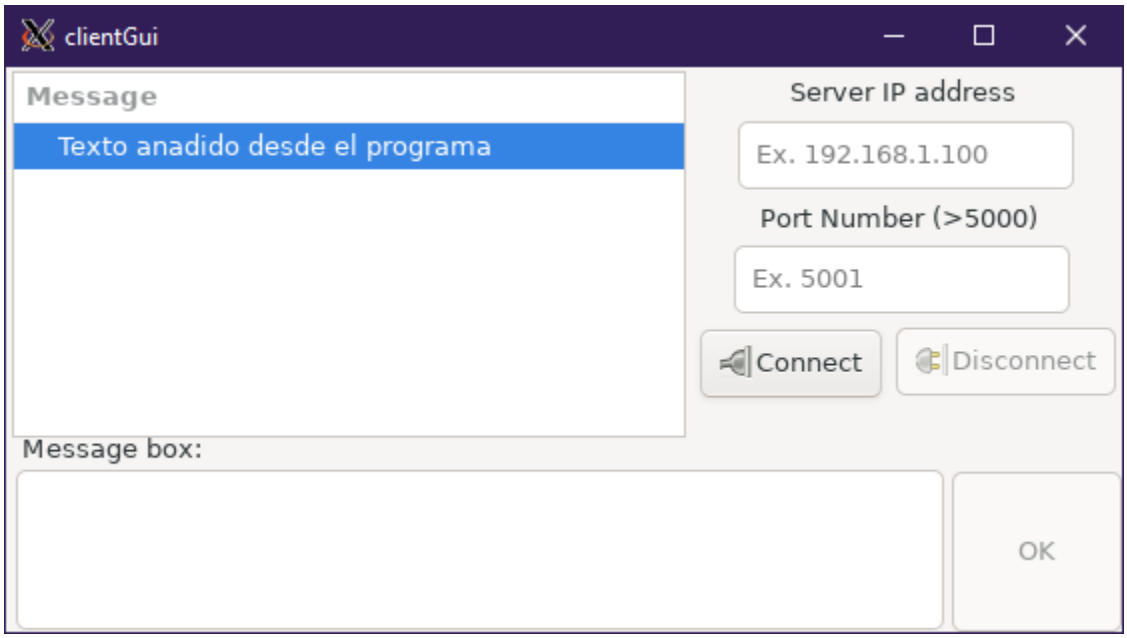


Image 4.7 Our new GUI, it does nothing right now, since we haven't added our signal handlers.

4.4 Sockets in C.

Socket programming is a way of connecting two nodes on a network to communicate with each other. One of the sockets listens, and the other tries to write to the other, they are usually in a client-server model [19]. Creating a socket in C consists in multiples steps, we will only revise some of them.

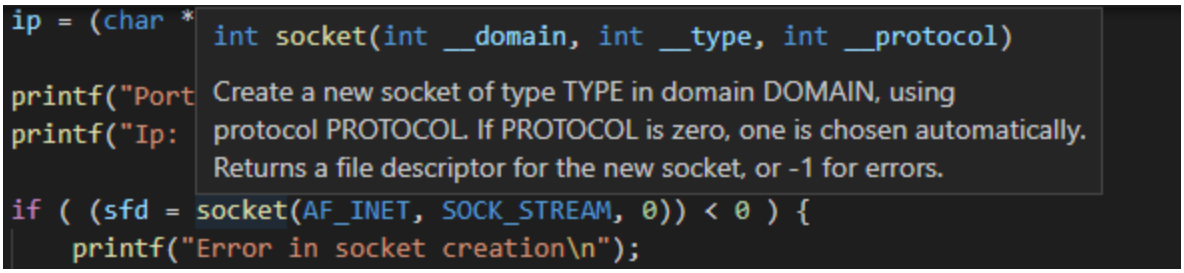


Image 4.8 Creation of a socket, AF_INET means IPv4 protocol, and SOCK_STREAM means bidirectional communication. [20]

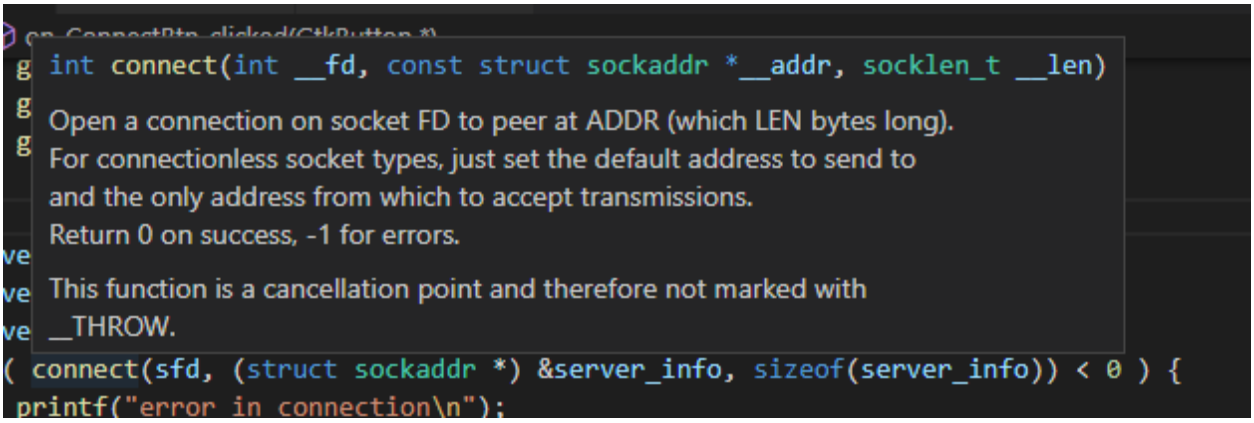


Image 4.9 Using our recently created socket, we connect to out peer at our server info. [21]

After these two steps, at least in the client side, our connection is being tried. A similar process is being defined in the server, creating a socket and then binding all of the server information together, in order to be able to accept connections.

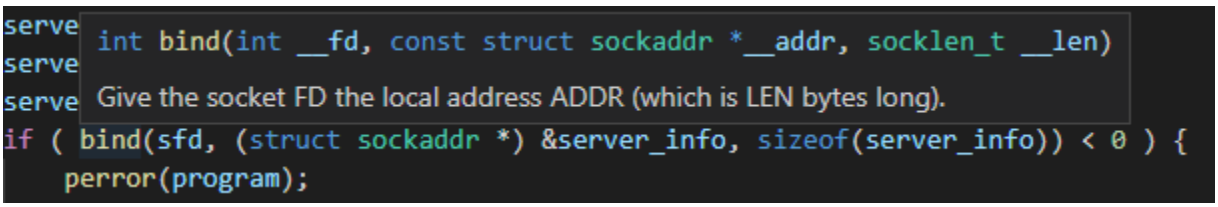


Image 4.10 Using our recently created socket, we bind our information together [22].

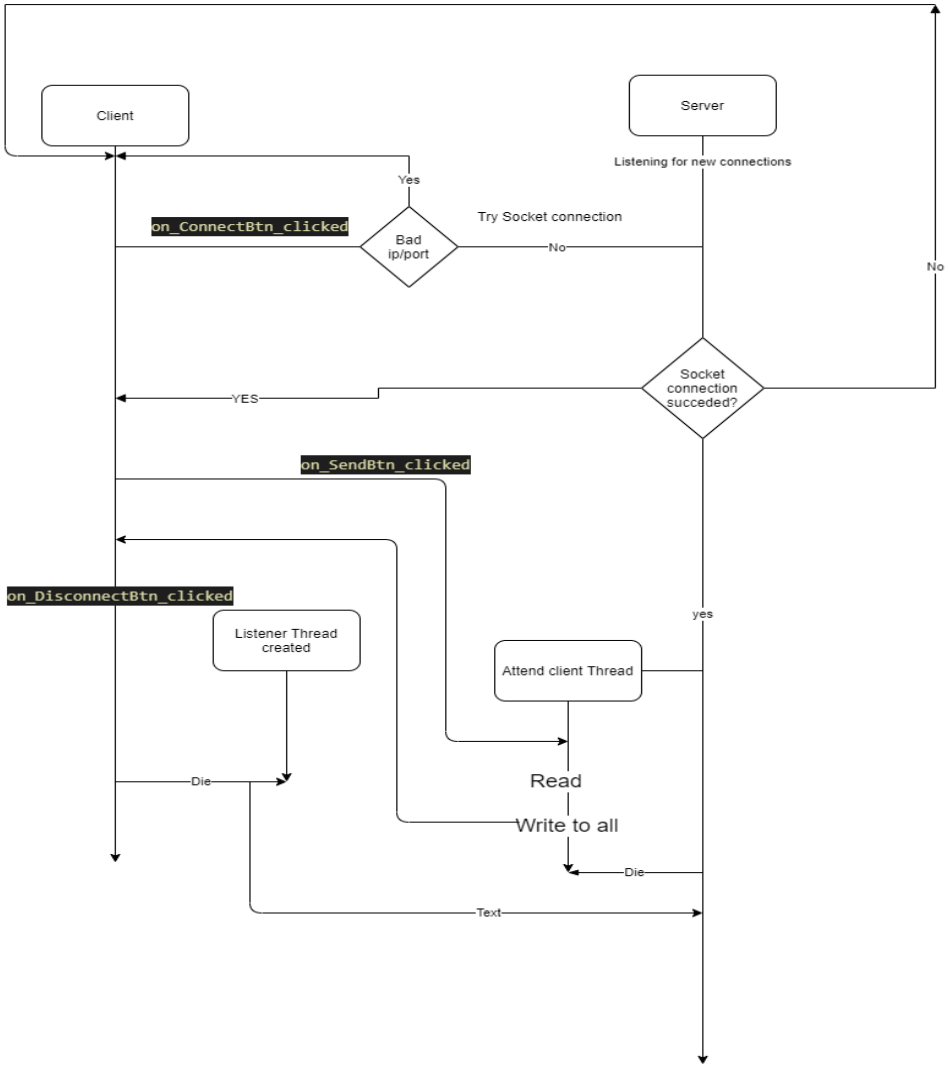
```
listen(sfd, 1000);
while (i<1000) {


    len = sizeof(client_info);
    if ( (nsfd = accept(sfd, (struct sockaddr *) &client_info, &len)) < 0 ) {
        perror(program);
        exit(-1);
    }
```

Image 4.11 Accepting connections in the server side. [23]

4.5 Using multi-threading and concurrency

In the next diagram you will be able the basic functionality of the chat server and client interactions.



```
pthread_t pthread_id;
connected = TRUE;
pthread_create(&pthread_id,NULL, clientListen,NULL);
```

```
pthread_create(&pthread_id[i],NULL, serves_client,NULL);
```

Image 4.12 & 4.13 Creating a thread for listening to every input from other clients, and creating a thread to handle every connection in the server.

```
void *serves_client() {
    int i=clients.i;
    int sock = clients.nsfd[i];
    unsigned char message[1024];

    message[1024]='\0';

    printf("\na:Connection Established: %i\n", clients.i);
    int j = 0;
    while (!strcmp(message,"Exit") == 0)
    {

        read(sock, &message, sizeof(message));
        printf("id: %i> %s\n", i, message);
        for (j = 0; j < totalClients; j++)
        {
            printf("\n\n\n\n\n\nJ!=I: %i> \n\n\n\n\n\n", j!=i);
            if(j!=i){
                printf("Imprimiré algo:\n");
                write(clients.nsfd[j],&message, sizeof(message));
                }
        }
    }

    printf("Connection Lost: id: %i\n", i);

    close(sock);
}
```

Image 4.14 Server-side client handler. That receives a new message, and iterate through the total of clients connected, and send them the message, if possible.

```
void *clientListen() {


    unsigned char answer[1024];
    //gtk_label_set_text(label,"text");
    // run the main loop to update the GUI and get it responsive:
    //while(gtk_events_pending()) gtk_main_iteration();

    // do some other computation...

    // huge computation in a loop:
    while(connected) {
        // do some computation...


        read(sfd, &answer, sizeof(answer));
        printf("-> %i\n", connected);

        gtk_tree_store_append(TreeStore,&iter,NULL);

        gtk_tree_store_set(TreeStore, &iter, 0, answer, -1);

        // update GUI and treat events from it:
        while(gtk_events_pending()) gtk_main_iteration();
    }


}
```

Image 4.15 Client side message listener, listens for anything the server sends to it. Also it updates the GUI every time a new message arrives.

5 Results:

My main goal was to create a concurrent server that can handle multiple connections to allow LAN communication, that purpose was accomplished.

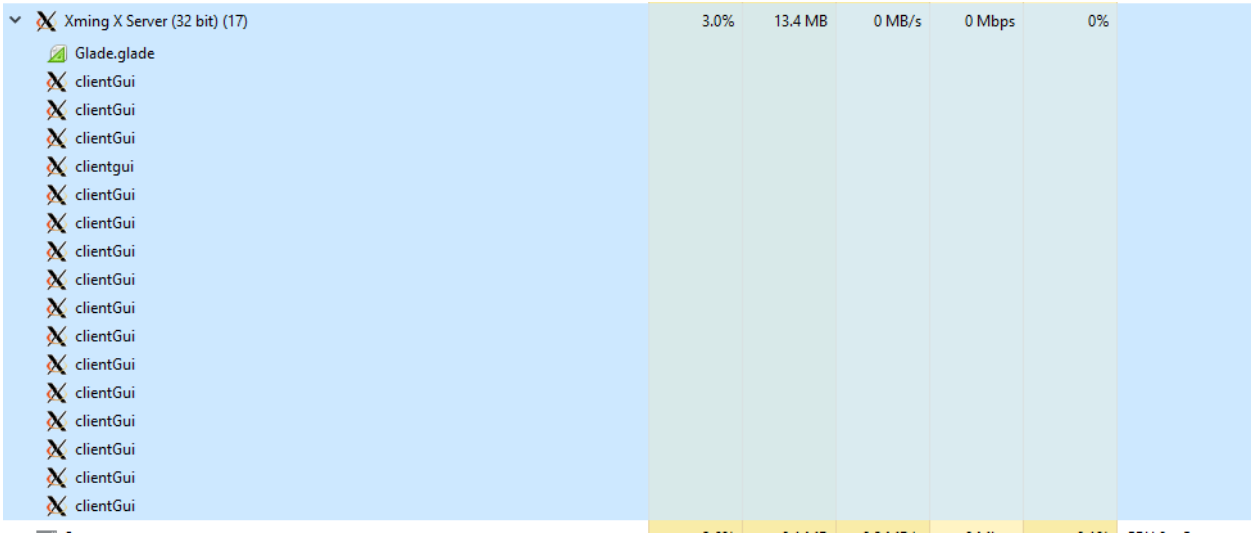


Image 5.1 Sixteen instances running at the same time. Taking 13.4MB of ram and 3.0% of my CPU capacity.

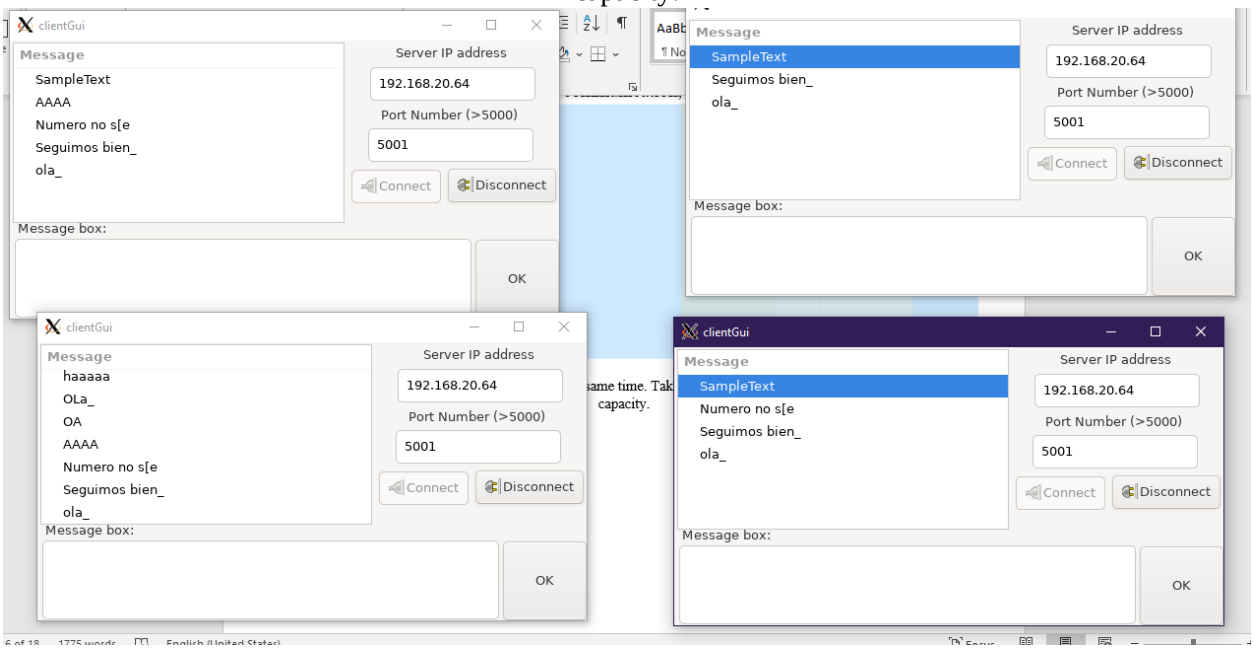


Image 5.2 Sixteen instances running at the same time. Output was still consistent, with older members of the group chat having more messages than the new ones.

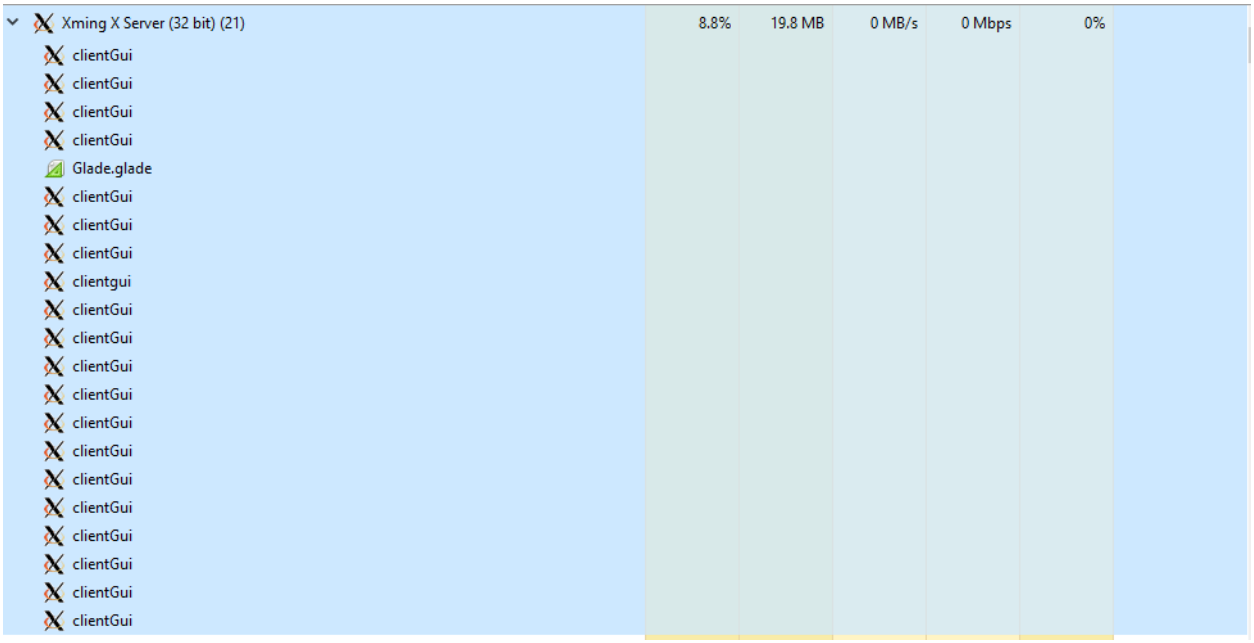


Image 5.2 twenty instances running at the same time. Output was still consistent.

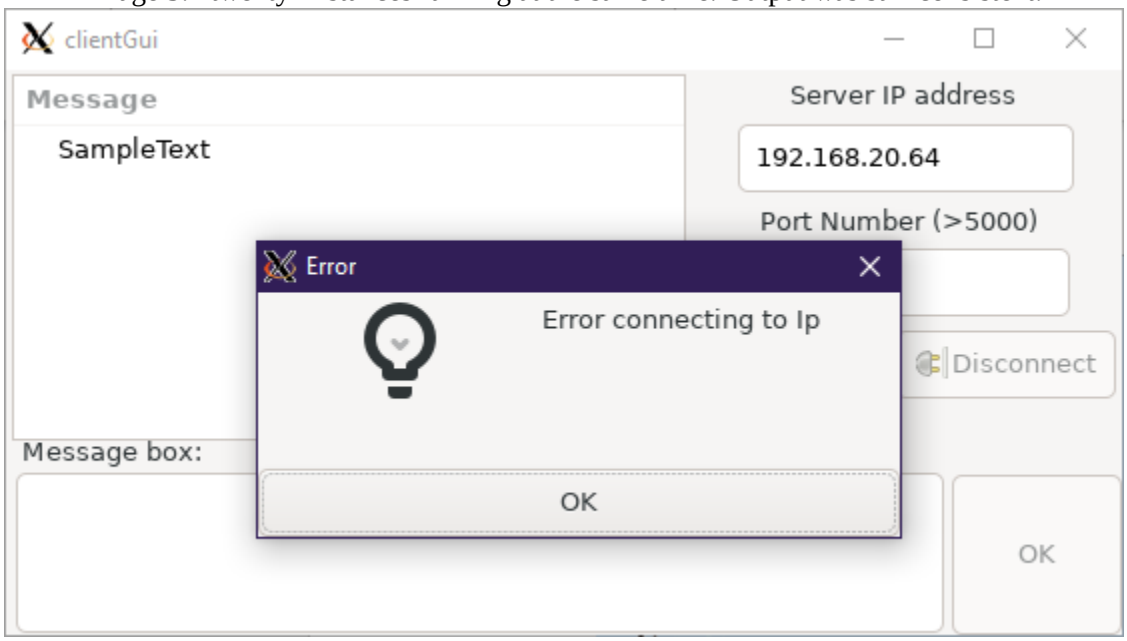


Image 5.2 twenty instances running at the same time. Error messages still consistent (wrong port).

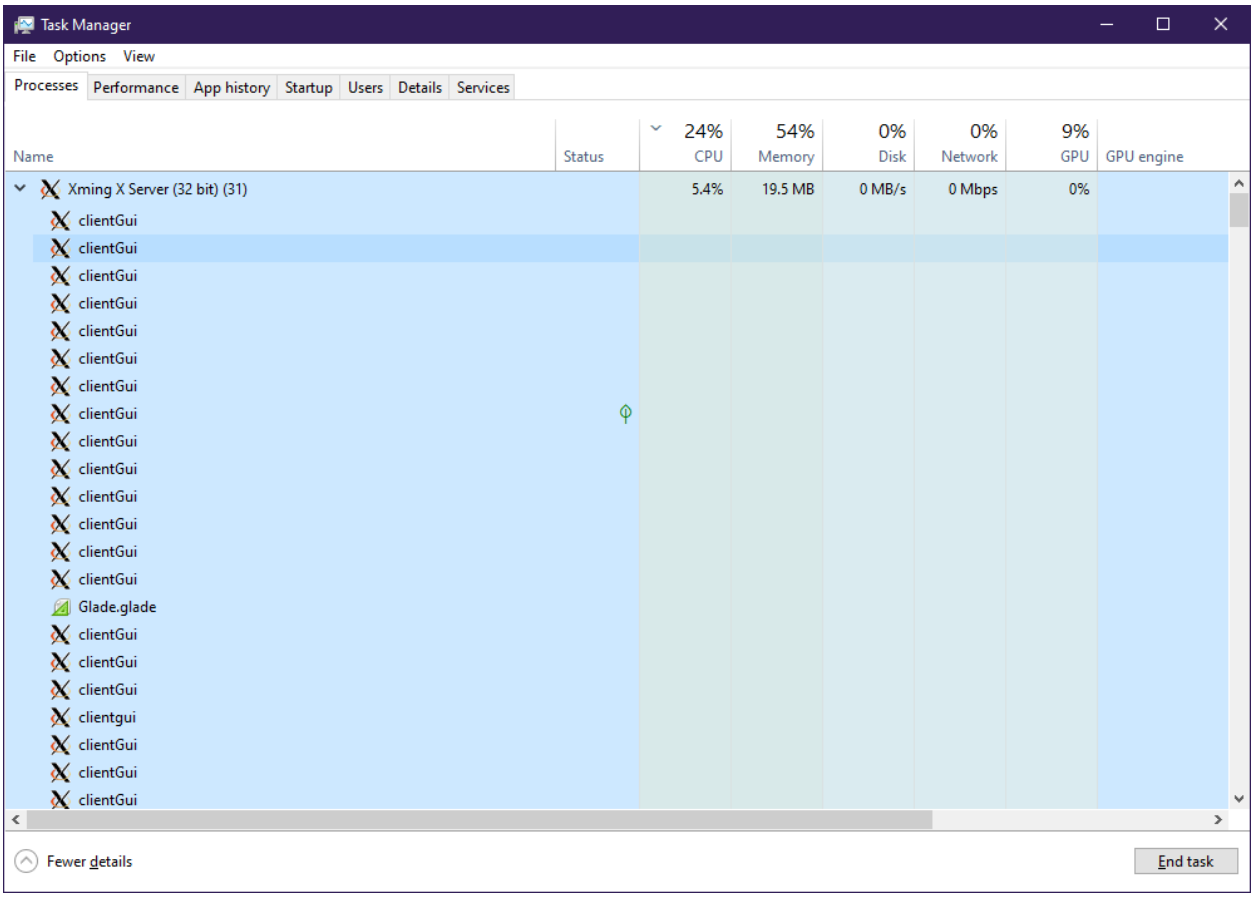


Image 5.2 Thirty instances running at the same time. Using about 24% of my CPU and 19.5 MB of RAM.

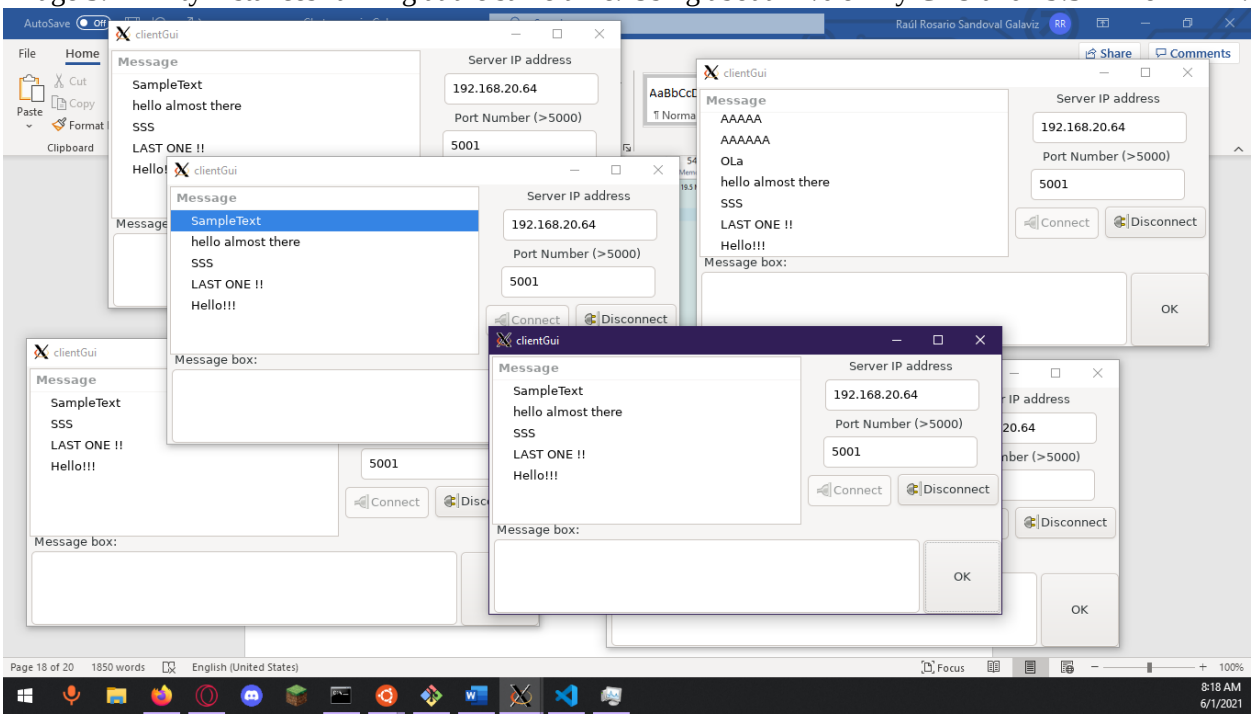


Image 5.2 Thirty instances running at the same time. Output is still consistent between each instance.

6 Conclusions

31 instances where run at the same time, every single one of them was able to still send, and receive messages, disconnect, and connect again, as expected. As we saw in the previous section, there could be an advantage that a small business in a rural area could take from an almost instant communication device such as this one [24]. During this project, I was able to understand concepts that were not completely clear from the get-go, like concurrency or parallelism, and I was able to identify ways of utilizing them to satisfy a necessity

References:

[1] https://www.whoson.com/our-two-cents/the-history-of-live-chat-software/
[2] https://www.appypie.com/why-messaging-apps-are-popular
[3] https://www.telcel.com/mundo_telcel/quienes-somos/corporativo/mapas-cobertura
[4] https://telmex.com/web/acerca-de-telmex/mapas-de-cobertura#
[5] http://www.inafed.gob.mx/work/enciclopedia/EMM26sonora/municipios/26033a.html
[6] https://web.mit.edu/6.005/www/fa14/classes/17-concurrency/
[7] https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html
[8] https://www.geeksforgeeks.org/difference-between-process-and-thread/
[9] https://en.wikipedia.org/wiki/LAN_messenger
[10] https://www.youtube.com/watch?v=Y1pgpn2gOSg
[11] https://glade.gnome.org
[12] https://en.wikipedia.org/wiki/GTK
[13] https://developer.gnome.org/gtk3/stable/ch02.html
[14] https://stackoverflow.com/questions/5450047/how-can-i-do-gui-programming-in-c
[15] https://en.wikipedia.org/wiki/Borland_Turbo_C
[16] https://help.gnome.org/users/glade/stable/introduction.html.en
[17] https://developer.gnome.org/gtk3/unstable/GtkBuilder.html#gtk-builder-new-from-file
[18] https://www.cs.uni.edu/~okane/Code/Glade%20Cookbook/01%20GtkButton/part1.c

[19] https://www.geeksforgeeks.org/socket-programming-cc/
[20] https://man7.org/linux/man-pages/man2/socket.2.html
[21] https://man7.org/linux/man-pages/man2/connect.2.html
[22] https://man7.org/linux/man-pages/man2/bind.2.html
[23] https://man7.org/linux/man-pages/man2/accept.2.html
[24] https://smallbusiness.chron.com/advantages-chat-room-70640.html